|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| logowydzialu | Instytut Informatyki Politechniki Śląskiej  Zespół Mikroinformatyki i Teorii Automatów Cyfrowych | | | logoii | |
| **Rok akademicki:** | **Rodzaj studiów\*: SSI/NSI/NSM** | **Przedmiot:** | | **Grupa** | **Sekcja** |
| **2018/2019** | **SSI** | **BIAI** | | **GKiO3** | **7** |
| **Autor:** | **Aleksandra Musiał (alekmus576@student.polsl.p)** | **Prowadzący**: | | **GB** | |
| **Autor:** | **Benedykt Waszkiewicz (benewas355@student.polsl.pl)** |
| ***Raport końcowy*** | | | | | |
| **Temat projektu:**  Classification of the genre of the film based on the description of the plot | | | | | |
| **Data oddania:**  **dd/mm/rrrr** | | |  | | |

1. Introduction

The aim of our project was to prepare a program capable of recognizing the genre of the movie based on the plot using neural network. We decided to implement the network in Python and use TensorFlow library.

1. Data set

We used dataset from www.kaggle.com which contains information about 35 000 movies. There are following columns in the set:

- release year,

- title,

- ethnicity,

- director,

- cast,

- genre,

- Wikipedia page,

- plot.

The release year varies from 1901 to 2017. As for genres, there are about 20 main genres but often multiple genres are assigned to one movie. Because of the assumptions we made at the beginning, we cut the dataset and left only columns with a genre and a plot.

1. Data preprocessing

From the original data set, the plot of the films and the corresponding genres are selected. First, the genres are standardized. Standardization in this case is the unification of the names of specific genres, because in the data set there are many small differences between the definition of the genres. For this reason, the names of genres, e.g. "sci-fi", "science-fiction", "science fiction", "sciencefiction." are changed to "science\_fiction". Next step is to replace the word abbreviations (e.g. "what's" with "what is"), punctuation marks and Wikipedia footnotes. Then unrepresentative genres are rejected, as well as those that could not be standardized and remain: "action", "adventure", "animation", "comedy", "crime", "drama", horror ", "musical ", "romance ", "science\_fiction" and "thriller ".

After rejecting unnecessary genres, the remaining rows are mixed up. This is done by randomly selecting a predetermined number of plot descriptions for a given genre. Mixing is done to get a more reliable result. After mixing the data, another standardization process follows. Because the data is written in words it should be convert to numbers. A dictionary [word, value] is created from all the words contained in the plot descriptions left in the data set, so that in the case of multiple occurrences of a given word, it has one numerical value. During this operation, insignificant words such as "are", "is", "about" are also removed, for this purpose the nltk library with the corpus of English words was used. The names of film genres are also converted into numbers. Standardization is followed by averaging the number of words in the story descriptions, because some of the descriptions have different amounts of them. Descriptions that have more words are truncated to a certain number, and those that have fewer words are copied from the beginning up to the specified number of words. At the end, all the rows are mixed up again so that the descriptions of the same genres stories does not occur in subsequent indexes. After mixing, the standardized data is saved to a csv file from which the neural network downloads them.

1. Implementation
   1. Keras

Keras is one of the leading high-level neural networks APIs. It is written in Python. The core data structure in Keras is the model which is available in two options – sequential and the Model class used with the functional API. We prepared the sequential one.

from tensorflow import keras

model = keras.Sequential()

* 1. Layers

We started building our neural network with the embedding layer. It is responsible for taking vocabulary in numerical form and looking up the embedding vector for each word-index. These vectors are learned as the model trains. It results in additional dimension in the output array – batch, sequence and embedding.

Later, we needed to be able to handle input of variable length. In order to do this, we added GlobalAveragePooling1D layer for global average pooling operation for temporal data.

Next layers which we appended were dense layers. The first one with bigger dimensionality of the output space and the second one with output dimension matching the number of the possible movie genres.

To avoid overfitting, we applied Dropout between two dense layers.

Because of lack of experience, we did not know how many layers will be the best for our project. That is why we started with two dense layers but while making experiments we decided that the neural network with only one dense layer is better in this case.

model.add(keras.layers.**Embedding**(input\_dim=vocabulary\_size,  
output\_dim= 512, input\_length=numberOfInputWords))

model.add(keras.layers.**GlobalAveragePooling1D**())

model.add(keras.layers.**Dense**(512, activation=tf.nn.tanh))

model.add(keras.layers.Dropout(0.3))

model.add(keras.layers.**Dense**(11, activation='softmax'))

* 1. Optimizer

Based on opinions in many articles about deep learning, we chose the **Adam** optimization algorithm as an optimizer in our project. The Adam is considered as one of the fastest algorithm.

* 1. Activation function

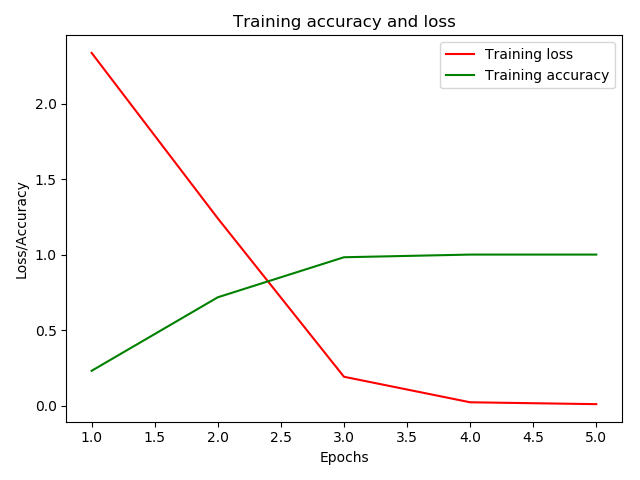
We made a lot of experiments to choose the best activation functions for dense layers. These two layers had different task to do, so we also matched different activation functions. **Softmax**  is frequently used in classifications – it turns numbers into probabilities. Pushing one result closer to 1 while another closer to 0, it sums outputs to 1 making great probability analysis. That is why we choose it.

Another activation function was **hyperbolic tangent** which gave us the best results while testing different options.

1. Results

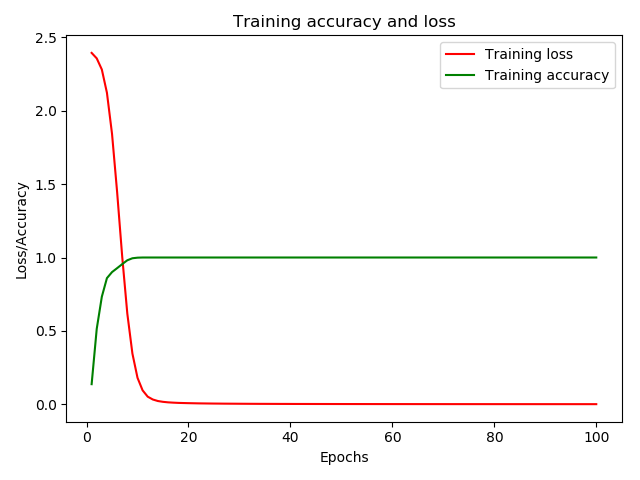
These are some of our results:

* with 2 dense layer
  + movies per genre: 250, words per plot: 500

a)

|  |  |
| --- | --- |
| Data set | 2750 |
| Training set | 1750 |
| Test set | 1000 |
| Epochs | 5 |
| Batch size | 25 |

Loss: 1.47, **Accuracy: 0.52**

b)

|  |  |
| --- | --- |
| Data set | 2750 |
| Training set | 1750 |
| Test set | 1000 |
| Epochs | 100 |
| Batch size | 250 |

Loss: 1.55, **Accuracy: 0.53**

* + movies per genre: 300, words per plot: 250

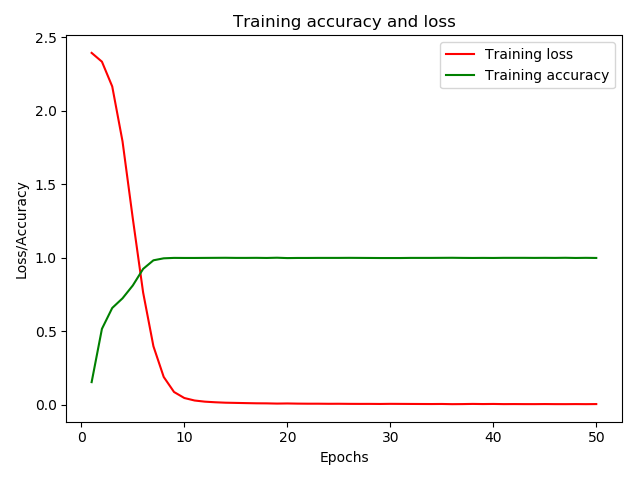
a)



|  |  |
| --- | --- |
| Data set | 3300 |
| Training set | 2500 |
| Test set | 800 |
| Epochs | 5 |
| Batch size | 25 |

Loss: 1.63, **Accuracy: 0.54**

b)



|  |  |
| --- | --- |
| Data set | 3000 |
| Training set | 2500 |
| Test set | 800 |
| Epochs | 50 |
| Batch size | 250 |

Loss: 1.64, **Accuracy: 0.54**

* + movies per genre: 400, words per plot: 200

a)



|  |  |
| --- | --- |
| Data set | 4400 |
| Training set | 3000 |
| Test set | 1400 |
| Epochs | 5 |
| Batch size | 25 |

Loss: 1.46, **Accuracy: 0.59**

* with 1 dense layer
  + movies per genre: 300, words per plot: 250

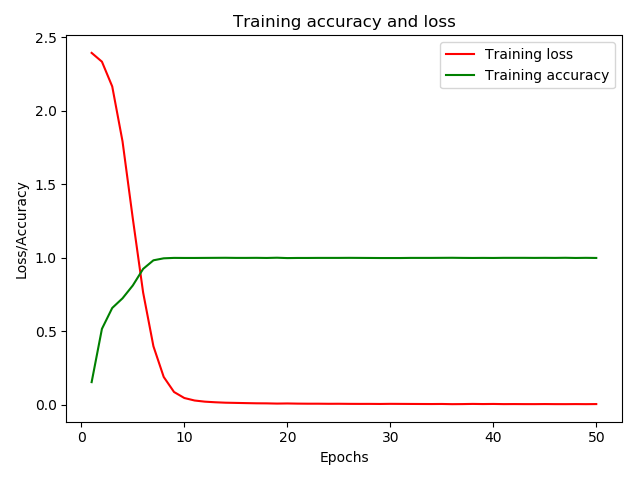
a)



|  |  |
| --- | --- |
| Data set | 3300 |
| Training set | 2500 |
| Test set | 800 |
| Epochs | 5 |
| Batch size | 25 |

Loss: 1.44, **Accuracy: 0.55**

b)



|  |  |
| --- | --- |
| Data set | 3300 |
| Training set | 2500 |
| Test set | 800 |
| Epochs | 50 |
| Batch size | 250 |

Loss: 1.64, **Accuracy: 0.54**

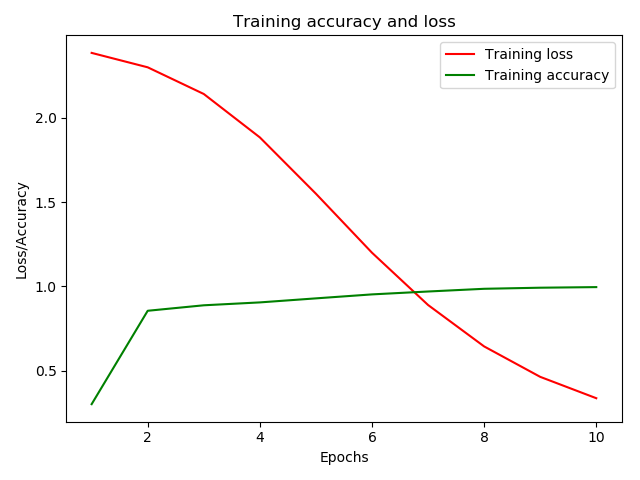
c)



|  |  |
| --- | --- |
| Data set | 3300 |
| Training set | 2800 |
| Test set | 500 |
| Epochs | 5 |
| Batch size | 25 |

Loss: 1.37, **Accuracy: 0.57**

d)

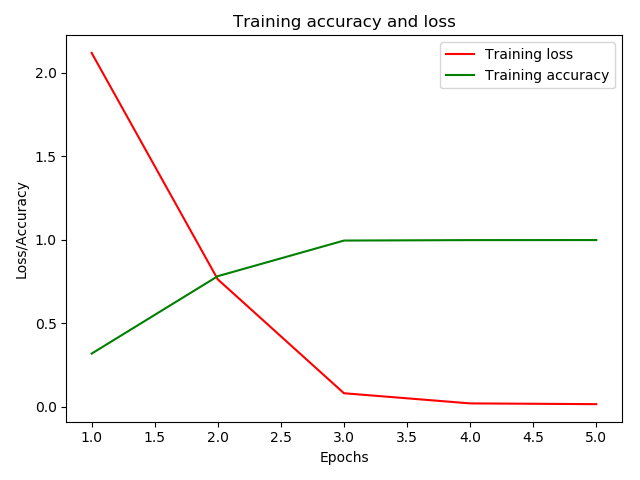


|  |  |
| --- | --- |
| Data set | 3300 |
| Training set | 3000 |
| Test set | 300 |
| Epochs | 10 |
| Batch size | 100 |

Loss: 1.44, **Accuracy: 0.57**

* + movies per genre: 400, words per plot: 200

d)



|  |  |
| --- | --- |
| Data set | 4400 |
| Training set | 3000 |
| Test set | 1400 |
| Epochs | 5 |
| Batch size | 25 |

Loss: 1.31, **Accuracy: 0.61**

1. Conclusions

As you can see in the previous part of this report, the best result which we have achieved was 61% of accuracy. The graphs show that accuracy rises when the amount of movies taken into consideration grows. When it comes to deciding on the best batch size and number of epochs for our neural network, you can see that training accuracy increases rapidly at the beginning and then remains stable. That is why we chose a few epochs and small batches. Another thing which has a big impact on the results is how much movies are prepared for each genre. Number of words in every plot is not so important because of different lengths of the text in the data set. Even if we increase number of words to 500, then in case of plot with 80 words the words will be repeated so it will not improve the value of training set. Moreover, while making some experiments, we noticed that one dense layer gives better results then two dense layers.

We believe that the key to improve the accuracy is bigger data set with consistent plots, without niche movies and obscure genres.

Additionally, we have used naïve Bayes classification to see if algorithm based off of Bayes’ Theorem can manage this task better than our neural network. This method is very popular in problems related to classification. We have achieved results for two genres – drama 88% and action 68%. We suppose that the difference in these results and results from the graphs stems from lack of experience in building machine learning model and imprecise data set.

To sum up, we have learned that there are many possibilities to use machine learning and every case should be considered independently. According to the graphs, not always more complex neural network means more efficient. For us, as the absolute beginners in machine learning, it was necessary to try a lot different options to build coherent and productive neural network.